ESD-TR-75-358

DRI File Copy
ESD ACCESSION LIST
DRI Call No. 84153
Copy No. 1 of 2 cys

NLS-SCHOLAR: MODIFICATIONS AND FIELD TESTING

Bolt, Beranek and Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

November 1975

Approved for Public Release;
Distribution Unlimited.

Prepared for and Sponsored by

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
HANSCOM AIR FORCE BASE, MA 01731

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY
1400 WILSON BOULEVARD
ARLINGTON, VA 22209
ARPA Order No. 2984

ADA021743

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U. S. Government.

LEGAL NOTICE

When U.S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

"This technical report has been reviewed and is approved for publication."

Sylvia R. Mayer

SYLVIA R. MAYER, GS-14
Project Scientist

FOR THE COMMANDER

Frank J. Emma

FRANK J. EMMA, Colonel, USAF
Director, Information Systems
Technology Applications Office
Deputy for Command & Management Systems

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ESD-TR-75-358 / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): NLS-SCHOLAR: MODIFICATIONS AND FIELD TESTING	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Mario C. Grignetti, Laura Gould, Catherine Hausmann, et al	8. CONTRACT OR GRANT NUMBER(s): F19628-75-C-0159, ARPA Order 2984
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Bolt, Beranek and Newman, Inc. 50 Moulton Street Cambridge, MA 02138	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62706E Program Element
11. CONTROLLING OFFICE NAME AND ADDRESS: Deputy for Command and Management Systems, Electronic Systems Division, Hanscom Air Force Base, MA 01731	12. REPORT DATE: November 1975
	13. NUMBER OF PAGES: 77
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Defense Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, VA 22209	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Artificial Intelligence, Computer Assisted Instruction, Natural Language Processing, Semantic Grammar, Semantic Network, Tutorial Supervision, On-Line Assistance, Question Answering

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

NLS-SCHOLAR is a prototype system that uses Artificial Intelligence techniques to teach computer-naive people how to use a powerful and complex editor. It represents a new kind of Computer Assisted Instruction (CAI) system that integrates systematic teaching with actual practice, i.e., one which can keep the user under tutorial supervision while allowing him to try out what he learns on the system he is learning about.

(over)

20. (cont)

NLS-SCHOLAR can also be used as an on-line help system outside the tutorial environment, in the course of a user's actual work. This capability of combining on-line assistance with training is an extension of the traditional notion of CAI.

The system is now operational. Limited but realistic testing revealed that the teachings of NLS-SCHOLAR are very effective, and that the system's performance as an on-line help facility needs improvement. Most of the problems encountered are very easy to fix.

The techniques used in NLS-SCHOLAR are general and can be applied to the teaching of a wide variety of computer related activities.

TABLE OF CONTENTS

SECTION I - INTRODUCTION

This is the Final Report on a six-month effort to improve and field test NLS-SCHOLAR [Grignetti 1975], a CAI system that employs Artificial Intelligence techniques to teach people how to use the BASE subsystem of NLS.*

This Report documents the changes made to the August 1974 version of NLS-SCHOLAR to prepare it for testing in the field, and documents the results and conclusions obtained from this testing. Since NLS-SCHOLAR was developed under a previous contract, this report is conceived as an "incremental" one that should be read in conjunction with the Final Report [Grignetti 1974] on our previous effort.

Overall Approach

NLS-SCHOLAR is oriented towards teaching NLS to naive users, such as secretaries, who have very limited experience with computer-based text editing systems. Therefore, its tutorial material is written assuming practically no knowledge of computer usage on the student's part; the necessary conceptual framework is built up from the most

----

*BASE is the powerful editor of the oN-Line System (NLS), an increasingly used text manipulation system developed by Douglas Engelbart and his co-workers at the Augmentation Research Center of Stanford Research Institute.

primitive notions, such as striking a key on a terminal keyboard.

The two basic pillars on which the system´s approach is founded are: a) interactiveness and mixed-initiative, and b) supervised practice of the procedural knowledge being taught.

Interactiveness and mixed-initiative are necessary so that the student doesn´t feel "caught" in a situation over which he has no control. The system is designed so that any time it is the student´s turn to type, he can ask questions himself (instead of just answering the questions posed by the system), or direct the system to perform certain actions for him (like executing NLS commands expressed in English).

Supervised practice is absolutely fundamental. Little knowledge about "how to do" things can be taught by mere descriptions; many procedures can only be taught by demonstration, and practice is essential. A supervised "hands on" environment is crucial to impressing newly acquired procedural knowledge in the student´s mind. NLS-SCHOLAR provides such an environment by requesting students to perform NLS editing tasks using (what appears to them to be) the very system they are being taught about, by remaining "aware" of what they are doing, and by commenting on their performance.

Objectives

Our ultimate goal is to develop NLS-SCHOLAR so that it can be used as an operational tool over the ARPA network, in support of the National Software Works (NSW) users. The specific objectives of the work described in this document were:

a) Expand and modify the NLS-SCHOLAR system as it existed at the end of its first year of development, incorporate features we perceived as needed, and correct known limitations

b) Test the newly developed system in a limited but realistic operational environment

c) Use the feedback and experience obtained in the field to evaluate the system and to formulate plans for the next stage of modification and expansion

These objectives have been achieved.

Outline

In Section II of the body of this report, we describe in detail the developmental work performed to achieve our objectives; in Section III we describe the results obtained during field testing of the present version of NLS-SCHOLAR.

Finally, in Section IV we present our conclusions and recommendations for further work.

SECTION II - DEVELOPMENTAL WORK

In this Section we describe the work accomplished to bring NLS-SCHOLAR to a state sufficiently stable and robust so that testing it operationally would yield meaningful results.

Overview

Our initial aim was to expand and improve NLS-SCHOLAR so that its tutorial material would cover most of the BASE subsystem of NLS. This entailed bolstering the system's question-answering abilities, expanding the task evaluation modules, and adding functions to the underlying LISP-NLS system (our own LISP implementation of the BASE subsystem of NLS).

In the course of our development work we brought up several versions of newly expanded and modified NLS-SCHOLAR systems, incorporating not only most of the features perceived as needed at the beginning, but many others as well. In fact, as work progressed and our experience running the system increased, we identified new requirements for both the short and the long term success of our system, and we performed work in addition to what was originally specified. This additional work included:

1) In order to provide the flexibility and modularity required to effect changes easily, we designed and implemented a new control structure that uses an implementation of the Bobrow/Wegbreit stack scheme for multiple environments ("spaghetti stacks") that is provided in the recently released LISP. [Bobrow 1973, INTERLISP 1975].

2) To increase the effectiveness of our tutorial material, we developed a prototype Agenda Language that allows us to write English-like lessons incorporating branching, remedial loops, quizzes, etc.

3) In order to provide a useful tutoring environment in spite of expected system limitations, we incorporated a fall-back mode wherein a human helper comes to the system´s rescue whenever the user requests it.

4) In order to make it practical and feasible to use systems such as ours in operational environments, we greatly improved the efficiency of NLS-SCHOLAR; not only is the output package 5 times faster, but the overall efficiency is twice as great.

By far the most significant of these advances was the design and implementation of a flexible control structure that uses the recently released "spaghetti" LISP. The

structure allows NLS-SCHOLAR to operate on multiple environments, making it possible for the various modules of the system (the English front end, the Quizzer, the Tutor Scheduler, NLS, the Task Monitor, and the Evaluator) to be handled like jobs in a time-sharing system. That is, processes request "the floor" as need arises, and gain access to the process queue with preassignable priorities. As a result of this improvement, the system now has the capability of back-tracking to abandoned contexts, of handling multiple tasks, and of coroutining.

We expanded the tutorials (the Primer) from the original three lessons to an introduction plus five lessons. The material covers usage of the legal combinations of the following NLS verbs and nouns:

a) Verbs: Load, Print, Insert, Delete, Create, Update, Jump, Substitute, Set, Reset, Show, Copy, Move, Transpose, Output, Help, and the one-character commands ´.´, ´/´, ´^´, ´\´, and <LF>.

b) Nouns: Character, Word, Text, Statement, Branch, Group, Plex, File and Rest.

Numerous questions, interspersed throughout the lessons and forming quizzes at the end, test the students´ comprehension of the instructional material. Over 100 supervised tasks and "tutor demonstrations" support our

claim that our users learn "by doing".

We developed a prototype Agenda Language that allows us to write these lessons in quasi-English format. (The lessons were all prepared using NLS and are in indented outline format.) The lessons contain not only tasks, demonstrations of actions, question-answering periods, and quizzes, but also branching and remedial loops. The new control structure allows us to design much more flexible lessons than before, ones that exhibit truly mixed user/system initiative. For example, one of the ways we can handle students´ mistakes is by means of "scratch actions": when a student makes a mistake, the system takes over and shows him what would happen if the the mistake were enacted. This resembles what a human tutor would do ("Here let me show you what would happen if you did what you propose") to show the effect of the mistake while at the same time protecting the student from the consequences of his actions.

In parallel with this work, our LISP implementation of the NLS BASE subsystem was augmented and updated, so as to support all the NLS commands mentioned above. We also sped it up considerably by using block compiling techniques.

Considerable work was done also on the English front end. In addition to questions, this module now handles all inputs from the student, including his answers to the "tutor´s" questions and his "directions" to the system. The

semantic network now contains 330 entries, covering the commands and NLS concepts which the simulator can handle and which the tutorials describe. The output package (the big CPU time gobbler in the previous system) was streamlined and speeded up by a factor of 5. In addition, the responses it produces are more personal and friendly.

Finally, in addition to the above, 1) we incorporated "stop" and "continue" facilities, so that users could proceed with the lessons at their own pace, 2) we began to provide users with some feedback on what went wrong when a question could not be answered by the system and, more importantly, 3) we offered students the help of a human user if they so required (the system looks for one of us, establishes a TENEX link, and allows us to come to the student's help and to the system's rescue). Contextual information (what the student has been up to) is preserved in LISP's history list and is available to us.

In short, we brought up a new NLS-SCHOLAR system that is very much better than the old one in terms of flexibility, modularity, capability, and efficiency.

In the remainder of this section we describe in detail the work performed in many of the areas alluded to above: the system's control structure, the tutorial material, the English front end, the human engineering features, and overall efficiency.

The Control Structure

The new control structure was designed with several goals in mind:

1) increasing the modularity of the system to make it more understandable and easily modifiable

2) facilitating interactions by a) making the "English understanding" portion of the system (ENGLISHEXEC) available at any time by a simple interrupt mechanism, and b) allowing the user to experiment with NLS at any time without destroying context

3) extending the capabilities of the tutorial material to permit branching and the conditional execution of arbitrary INTERLISP functions to perform needed actions.

The basic idea underlying the control structure is simple. The system continually evaluates the priorities of several alternative goals, which include ones specified by the user and ones set by the author of the tutorial material. Goals with lower priority are postponed, and the highest priority goal is executed. Some goals, such as "presenting all the tutorial material in a useful order", are complex and may continue over a long period of time. To facilitate the description of complex, long lasting goals, each goal is represented by a "process", a collection of INTERLISP procedures which when executed will achieve the

goal.

Because the spaghetti-stack control structure of INTERLISP permits any process to be interrupted at an arbitrary point without losing the context of the computation, complex goals can be represented by processes which work through a set of sub-goals from beginning to end without interruption. A process representing such an extended goal may be interrupted and temporarily suspended to allow other goals to be met. This permits the overall system to "stop in its tracks" and interact with the student when the student wants help, not just when the system decides to pay attention. In this way the control structure makes it possible for us to design a truly "mixed-initiative" system, rather than representing a single-minded tutor, since the various goals of the tutor may be easily interrupted and suspended to allow the student to request actions, ask questions, and experiment with NLS.

The overall control of the system is based in a simple "monitor" which acts much like a time-sharing monitor - it has a set of suspended processes representing pending priorities which must be evaluated, and it chooses the highest priority process and permits it to run.

At any time there may be several pending goals in the system, represented by suspended processes. These goals are chosen from the set:

a) listening for user commands, questions and answers in English (ENGLISHEXEC)

b) deciding what tutorial material to give next

c) presenting a tutorial unit

d) presenting a question

e) waiting to evaluate the answer of a previous question

f) running a student through an NLS task

g) providing an experimental NLS environment requested by the user

The priority evaluation is implemented primarily by a stack, but it is made potentially general by having the monitor evaluate a priority setting process associated with each runnable process, and using that to modify priorities. In addition, the stack of processes is easily accessible to running processes, and thus processes can (and do) add and delete processes on the stack.

In addition, by making use of the user-defined interrupt character facility and the features available in the new "spaghetti stack" version of INTERLISP, it is possible for the user to interrupt any process, save its context completely, and start up a copy of the ENGLISHEXEC which can answer general NLS factual questions, or start up a safe NLS environment on which to experiment without affecting the current NLS environment. This enables students to try out risky procedures without fearing the

consequences of potentially costly mistakes.

The spaghetti stack features permit the entire context and state of a complicated (perhaps recursive) process to be saved, to be run later or examined by other programs. This has been used to implement a "coroutine package" which greatly facilitated writing simple, easy to understand modules.

An example of this is the "question posing and evaluation module". This module is run having as arguments a question to be posed to the student, and evaluation procedures for possible answers. It would be easy to write if it were expected simply to pose a question and to interpret the next student input as an answer. However, we wished to allow the student to interact with the ENGLISHEXEC once the question is posed, by asking questions or typing in commands if he desires. Thus answer evaluation must be held in abeyance until the student actually types in an answer.

With the coroutine package this is simple - the question-posing module calls a coroutine which puts the question-posing-module on the stack with the evaluation section to be run next, puts an ENGLISHEXEC process on the top of the stack, and then cedes control to the monitor. When the ENGLISHEXEC recognizes an input as an answer, it removes itself from the stack and calls the question-posing module as a coroutine. To the question-posing module the

net result is that the student's answer is made available as if from a subroutine. While this could have been done with subroutines, the coroutine technique substantially simplifies the state of the system during the period after the question is posed.

Tutorial Material

The tutorial material has been expanded considerably since November, 1974, and now consists of five lessons rather than three. These lessons describe the BASE subsystem of the teletype-oriented* version of NLS as it appeared in March, 1975; they are written specifically for naive users with no previous knowledge of NLS and (perhaps) no previous acquaintance with terminals or computer systems.

New Text - To facilitate the initiation of these naive users into the mysteries of computer-assisted instruction, an interactive introduction has been written which gives a brief description of the goals of the system and explains the use of <CR> to terminate commands, <CTRL-A> and <CTRL-X> for line editing, <CTRL-T> to determine the state of the job, and <CTRL-H> to get the attention of the "tutor"

---

*We use the term teletype to denote generically a hard-copy terminal, as opposed to a display terminal.

(ENGLISHEXEC). This introduction supplants and surpasses the instruction sheet handout which was used for this purpose previously.

The five lessons differ in both content and structure from their predecessors. Revision of the content of the original three chapters was made necessary by changes in the NLS syntax and in a few NLS commands. The material was extended to provide more examples and to present commands not previously covered. These new commands include Print File, Print Rest, print the context of the CM (/), Reset Viewspecs, and Output Sequential File for producing a text file which can be listed on a line printer. A brief description of the Help command is given at the end of the last lesson so that the "graduating" student will know how to make use of this facility when he uses NLS without tutorial supervision. A small, self-contained help data file about viewspecs has been provided for practice with the use of this command.

<u>Branching</u> - These changes in content, however, are of much less significance than the increased freedom granted to the student by the new control structure, and to the author of the tutorial material by the introduction of branches and remedial loops. The ability to use branches means that the order and the content of what is being presented to the student can be made dependent on his choices or on his

performance. The addition of these facilities transformed the task of providing the tutorial material from writing a text (the Primer) to designing a programmed instruction course.

From the students´ point of view, each lesson (and the introduction) is composed, as before, of short sections of text which are printed at the terminal. At the end of such presentations, the student is given the opportunity to request more text, to ask any number of questions, or to practice with NLS using any commands that he chooses.

Tasks - Some text sections are followed by tasks which the student is asked to perform. In the course of doing a task, the student may use <CTRL-H> to get the "tutor´s" attention; he may then ask questions, practice with NLS to see the effect of a command, ask that he be allowed to restart the task, or ask that the task be done for him. If the student performs the task, his work is evaluated and helpful comment or criticism is provided. If his work is unsatisfactory he may be asked to do the task again, either wholly or partially.

Questions - Some text sections are followed by questions for the student to answer. In the course of trying to answer the question he may ask questions himself, or practice with NLS in an attempt to determine the answer.

A set of questions (a quiz) has been placed at the end of the introduction and of the first two lessons so that the student may have this additional method of assessing his progress. Answers are evaluated and appropriate responses made. Considerable latitude is provided in the judging of answers so that the student is not constrained to a particular form. For example, the question "What is the statement number of the origin statement" may be answered 0, statement 0, or zero; all are equally correct. In cases in which an answer has several parts, missing information is often supplied in the evaluation.

Answers - The handling of students' answers is made easy by the use of answer predicates. A sequence of these predicates can be written by the author after each question; the predicates are then tested one after the other until one of them succeeds. They operate in two steps: the first one provides for extracting expected words, for testing those words in various ways, and for filtering out irrelevant parts of the answer; the second step is some action which is undertaken or not, depending on the outcome of the first. These actions generally consist of some text being printed followed by an optional branching instruction.

English Front End

The English front end handles all language input from the student. It therefore must be powerful enough to distinguish between commands, ("Start lesson 5", "Delete branch 2"); queries, ("How do I print the whole file?"); and replies to tutor-generated questions ("The statements are 4A and 4B"). We decided to use the notion of a semantic grammar [Burton 1975] with two important additions, namely instantiation of variables and Case assignments [Fillmore 1968]. These two processes will be described later.

The key notion underlying the semantic grammar approach to parsing is the replacement of the search for syntactic constructs by a search for semantic ones. Parsing a student's request in this way yields its meaning <u>directly</u>, i.e., it produces an <u>executable</u> retrieval formula that prescribes a search in the system's "data base" (the semantic network plus the user's work space). The search can then be carried out and the results used to synthesize an answer to the request. Notice that in such a parsing process there are no separate syntactic and semantic phases (as there are in systems like the LUNAR parser [Woods 1972]).

<u>The Parsing Process</u>- The parsing process begins with a prescan of the student's input. Abbreviations are expanded, synonyms are recognized and rewritten into a canonical form, and compounds are collected into one word. These processes

ease the work of the parser itself by cutting down on the number of alternatives that must be considered.

After the input is prescanned, an attempt is made to parse it using an embodiment of the grammar described in BNF in Figure 1. Each non-terminal node of the grammar is a semantic category which takes into account all the predicted ways of expressing it. Each semantic category is embodied in a LISP function that tests the input string (or a substring of it) to determine if it belongs to the category. If successful, the function returns a value which condenses the "meaning" of the string.

The top level rule is <REQUEST>, which can be realized by four semantic categories: <DIRECTIVE>, <QUESTION>, <NLS/ACTION/REQ>, and <ANSWER>. This means that an input from the user (a request) can be either a directive, a question, an NLS command expressed in English, or an answer to a question asked previously by the system. Each alternative is tried sequentially until one succeeds. If none succeed, an error message is typed to the student ("I didn´t understand that. Please rephrase.") A good way to describe the parsing process is by example. We shall follow the parsing of the request "What command prints the next statement?" (see Figure 1).

```
<REQUEST>:= <DIRECTIVE>
            <QUESTION>
            <NLS/ACTION/REQ>
            <ANSWER>

<DIRECTIVE>:= ? !  CHECK ! PLAY ! RESTART ! GO ! HELP ! STOP

<QUESTION>:= <DEFINE/REQ>
             <WHATIS/REQ>
             <CONTENT/REQ>
             <PARTS-IN-PART/REQ>
             <PARTS-IN-LEVEL/REQ>
             <PROCEDURE/REQ>
             <TYPE/REQ>
             <INSTR/REQ>
             <POSITION/REQ>

<NLS/ACTION/REQ>:= <ACTION/SPEC>

<ANSWER>:= <THE-ANSWER>
           <DONT-KNOW-ANSWER>
           <LIST-ANSWER>

<THE-ANSWER>:= [THE THEY IT] [IS ARE]

<DONT-KNOW-ANSWER>:= TELL\ME ! I DON´T KNOW

<LIST-ANSWER>:= a list that doesn´t begin with a <VERB>
                or a question word like What, Is, Why, etc.

<DEFINE/REQ>:= [DEFINE DESCRIBE] <NOUN>
               WHAT DOES <NOUN> [DO MEAN STAND\FOR]
               HOW DOES <NOUN> WORK

<WHATIS/REQ>:= WHAT\IS*
                  [PURPOSE\OF <NOUN>
                   CONTENT\OF <STR+ADDR>
                   LEVEL\OF <STR+ADDR>
                   PROCEDURE\FOR <ACTION/SPEC>
                   ADDRESS\OF <STR+ADDR>
                   EXAMPLES\OF <NOUN>
                   EXAMPLE\OF <NOUN>
                   DEFINITION\OF <NOUN>
                   <CURRENT/PART>
                   <STR+ADDR>
                   <NOUN>]
                *Also SHOW\ME TELL\ME GIVE\ME TELL\ME\ABOUT
                 WHAT\ARE

<CONTENT/REQ>:= WHAT <STRUCTURAL> CONTAINS <STRING>
```

Figure 1. BNF description of the grammar.

```
<PARTS-IN-PART/REQ>:= WHAT <STRUCTURAL> ARE IN <FILE/PART>
WHAT ARE <STRUCTURAL> IN <FILE/PART>

<PARTS-IN-LEVEL/REQ>:= WHAT <STRUCTURAL> ARE <LEVEL/PART>

<PROCEDURE/REQ>:= [HOW\DO\I SHOW\ME\HOW\TO TELL\ME\ABOUT] <ACTION/SPEC>

<TYPE/REQ>:= WHAT CAN I TYPE AFTER [<VERB> <STRING> <PROMPT>]
             WHAT CAN FOLLOW [<VERB> <STRING> <PROMPT>]

<INSTR/REQ>:= WHAT (COMMAND) <ACTION/SPEC>

<POSITION/REQ>:= WHERE AM I
                 WHERE IS/ARE <STR+ADDR>

<ACTION/SPEC>:= <VERB> [<OBJ>]

<VERB>:= word whose part of speech is Verb

<OBJ>:= [<RELATIONAL>] [<NOUN/PHRASE>] [<OBJ>]

<RELATIONAL>:= words like NEXT\TO FROM AT TO, etc.

<NOUN/PHRASE>:= <TASK>
                <STR+ADDR>
                <FILE>
                <NOUN>

<TASK>:= TASK <NUMBER>

<STR+ADDR>:= <FILE/PART>
             THE <STRUCTURAL> <STRING>
             THE <TEXTUAL> <STRING>
             <CURRENT/PART>
             <STRING>

<FILE>:= (NLS\FILE) [BREAKFAST DINNER MYBREAKFAST]

<NOUN>:= any word whose part of speech is Noun

<NUMBER>:= a number

<FILE/PART>:= STATEMENT\0
              GROUP <ADDRESS> <ADDRESS>
              [STATEMENT STATEMENT\NUMBER BRANCH PLEX] <ADDRESS>

<ADDRESS>:= a word whose first character is a number

<STRUCTURAL>:= STATEMENT ! BRANCH ! GROUP ! PLEX

<STRING>:= a string delineated by double quotes
```

Figure 1 (cont)

```
<TEXTUAL>:= WORD ! CHARACTER ! TEXT

<CURRENT/PART>:= CURRENT\NLS\COMMAND
                 CURRENT\VIEWSPECS
                 CURRENT\STATEMENT
                 NEXT\STATEMENT
                 BACK\STATEMENT
                 CURRENT\ADDRESS
                 POSITION\OF\THE\CM
                 CURRENT\STATEMENT\NUMBER
                 CURRENT\FILE
```

Figure 1 (cont)

In the prescan the words "next statement" are recognized as a compound word or concept and are rewritten as next\statement. Starting with the grammar rule <REQUEST>, the first check is to see if the sentence is a <DIRECTIVE>. It fails and the next one is tried, <QUESTION>. The first seven realizations of the rule fail; but <INSTR/REQ> succeeds with "What" being followed optionally by the word "command", followed by an <ACTION/SPEC>. <ACTION/SPEC> succeeds, since "print the next\statement" is indeed an action specification. <ACTION/SPEC> returns as its value (remember it is a LISP function) an expression that is the "meaning" of the action specification:

```
((VRB PRINT) (OBJ NEXT\STATEMENT))
```

This says that the action is represented by the verb "print" and the object of the action is "next\statement". In turn, <INSTR/REQ> returns:

```
(QFIND/INSTR ((VRB PRINT) (OBJ NEXT\STATEMENT)))
```

which represents the "meaning" of the sentence. At this point the parsing phase is complete.

To find the correct answer, this "meaning" is executed as a LISP expression. (QFIND/INSTR is the function and VRB and OBJ are its arguments). The function QFIND/INSTR first checks to see if there is an OBJ. If there is one, it looks

under the OBJ´s data base entry for a section of data base beginning with the VRB. If that search fails, a general reply is given by finding all instruments (commands) under the VRB print and printing out the procedure for using each one. In this way, most of the knowledge the data base contains about printing would be given to the student. The belief is that a complete description is better than a simple "I don´t know". Among all these procedures, the student may find the one he was looking for.

In our example, the search for the VRB under the OBJ succeeds (see figure 2).

Figure 2

```
NEXT\STATEMENT
   (PRINT (I 2) (AGENT NIL USER)
                (OBJ NIL NEXT\STATEMENT)
                (INSTR NIL <LF>\COMMAND))
```

The English output routines take the piece of data base and form the English sentence:

YOU PRINT THE NEXT STATEMENT USING THE <LF> COMMAND.

<u>Fuzziness</u> - The parser allows for fuzziness; that is, it is able to skip over words in a controlled way in order to achieve a parse. The hope is that these words are noise words or at least that they can be skipped over and still permit a parse that is not far from the real meaning of the

request. The problem is that in some cases fuzziness leads to a completely different meaning. For example, consider the sentence "What are the default viewspecs?". In pushing for an object, let's say the parser doesn't recognize the word "default". Fuzziness would allow the parser to skip over this word. It recognizes "viewspecs", and in effect parses the sentence as "What are the viewspecs?". Applying fuzziness techniques well is a very tricky business!

<u>Instantiation</u> <u>of</u> <u>Variables</u> - An effort was made to see what it would take to build an English front end for NLS that would allow the student to express NLS commands in English. The added bonus from this research was the ability to answer with greater precision questions that dealt with more specific information than the data base <u>explicitly</u> contains. An example is the sentence "How do I delete a structure unit" versus the more specific request "How do I delete plex 2?" This ability was achieved by adding to the data base a new construct: instantiation variables that may get set during parsing and, if so, will be used in place of the general term -- otherwise the more general term is used. For example, in the data base entry for DELETE\COMMAND, the string $INS appears 3 times. Each time it is followed by a variable name, (XOBJ, XOBJSTR, or XADDSTR) and then followed by a regular piece of SCHOLAR data base (see Figure 3).

Figure 3

```
DELETE\COMMAND
  [PURPOSE (I 2) (DELETE NIL
     (AGENT NIL USER)
     (OBJ NIL ($INS XOBJ ($EOR (NAME NIL (OF NIL STRUCTURE\UNIT))
                                (NAME NIL (OF NIL STRING\UNIT)))))
     (INSTR NIL DELETE\COMMAND)
     (PROCEDURE NIL (TYPE NIL
            (AGENT NIL USER)
            (OBJ NIL ($SEQ "DELETE "
                           [$INS XOBJSTR
                              ($EOR (NAME NIL (OF NIL STRUCTURE\UNIT))
                                    (NAME NIL (OF NIL STRING\UNIT]
                           ($INS XADDSTR ADDRESS)
                           <CR> <CR>]
```

In processing "How do I delete a structure unit" none of the instantiation variables is set and so a general response is given:

YOU DELETE A STRUCTURE UNIT OR A STRING UNIT USING THE DELETE COMMAND.
PROCEDURE: YOU TYPE ´DELETE ´, FOLLOWED BY THE NAME OF A STRUCTURE UNIT OR THE NAME OF A STRING UNIT, THE ADDRESS, <CR>, AND <CR>.

In processing "How do I delete plex 2", <u>all</u> of the variables are set during parsing so a very specific reply can be given:

YOU DELETE PLEX 2 USING THE DELETE COMMAND.
PROCEDURE: YOU TYPE ´DELETE ´ FOLLOWED BY ´PLEX ´, ´2´, <CR>, AND <CR>.

Now, not only can the question be answered, but it can be turned into a command to NLS to perform the action "Delete plex 2" on a copy of the user´s file. It parses as

an <NLS/ACTION/REQ>. The form returned from the parse is

```
(QDO/PROCEDURE (VRB DELETE)
               (OBJ PLEX (ADDR 2))
```

QDO/PROCEDURE is a function which first retrieves the appropriate piece of data base and checks to see if all the instantiation variables in this piece are filled in. It then calls LISP-NLS, handing down to it the legal command sequence. (If all the instantiation variables were not set during the parse, a reply is generated telling the student what is missing.) Using a copy of the student´s current file, LISP-NLS executes the command sequence:

```
     BASE C: Delete C: Plex (at) A: 2;
     OK: ;
```

<u>Further uses of LISP-NLS to answer questions</u> - We have just described one use of LISP-NLS: responding to an English request to have NLS perform a command. A second use is to respond to queries like "Where am I now" and "What is the address of the statement containing "PRIME"?" These kinds of requests imply that at least one NLS command be performed. In the first case the answer can be found by performing the "." command; in the second by performing a series of commands - Jump Address 0, Jump Address "PRIME", then "." to get the current address.

Human Engineering Features

In order to make NLS-SCHOLAR an easy and pleasant system to use, we strived to endow it with a number of human engineering features that will be described next.

Stop and resume. - Sessions with NLS-SCHOLAR have natural breaking points, such as lesson boundaries or large topic changes, at which it is convenient and even desirable for a user to quit. Having stopped at one of these system-provided breaks, the user can resume the lesson at a later time by asking the system something like "start lesson 3 now, please". Often, however, users find the time between these natural breaks to be too long, either because their own performance has required a longer time than average, or because something else demands their attention. We have provided the system with the necessary mechanism for allowing those users to stop the lesson at any time, in whatever situation they may find themselves: in the middle of a lesson, performing a task, answering a question, or even working with NLS doing their own thing. All they have to do is get the attention of the "tutor" (by typing <CTRL-H>) and then tell it they want to stop. The system responds by asking the user to confirm his request and to indicate if he intends to continue at a later time. If both answers are affirmative, the system writes out a file (a LISP SYSOUT file) in the user's directory. When the user

comes back, the system reminds him of the existence of a suspended work session; if the user wants to, he can continue exactly where he left off by simply typing RESUME (which causes a LISP SYSIN). This feature was very sorely needed and was used by almost all those involved in the field testing.

<u>Getting help from an expert</u>. - Since we did not expect our system to be able to comprehend all user requests and to always provide useful answers, we endowed NLS-SCHOLAR with a feature that allows a human expert to come smoothly to the system's rescue when the system fails. This facility operates as follows. Let's suppose that a user is in the middle of a task, asks a question whose answer is badly needed, and the system either fails to understand his question or gives him an unsatisfactory answer. If he asks for help at this point, the system will seek a logged-in human expert, establish a link, and report the failure to the expert. If it isn't possible for the expert to provide the answer solely on the basis of an isolated question, he can examine a history list maintained by the system. This list is a record of previous interactions between user and system which provides the context the expert often needs to answer a question appropriately.

The main reason for the incorporation of this facility was to allow our students to utilize lesson time more

effectively; we wanted their experience using NLS-SCHOLAR to be a profitable one in spite of the system's limitations, and we hoped the facility would minimize frustration and unnecessary breaks. In spite of our hopes, the facility was hardly used at all: only one of our users ever attempted to take advantage of it, but unfortunately no expert was logged-in at the time help was sought.

Question mark. - Given the great flexibility of the control structure, the student may well be confused as to what to do when he gets the "tutor's" attention. A question mark facility was implemented to help users remember what they could request the system to do for them. When the student types a "?", the system responds with a list of one-word commands which may be used to initiate actions, such as starting a lesson, restarting a task, stopping a lesson, resuming it, summoning help, calling NLS, etc. These actions are not necessarily invoked specifically by their associated command; rather, it is the combination of command and situation that decides which action will be undertaken. Thus, if a user types "continue", several things may happen: a) if he was in the middle of a lesson, the lesson continues; b) if he was performing a task, he goes back to the task's environment; c) if he just entered NLS-SCHOLAR and there is a stopped lesson under his name, the lesson is resumed; d) if he was working with NLS doing his own thing,

he is returned where he left off.

Efficiency - The newly brought up NLS-SCHOLAR system is remarkably more efficient, in terms of CPU utilization, than its predecessor: it takes about 3 minutes of CPU time, on the average, per lesson hour. This efficiency measure applies to a lightly loaded TENEX system; under these circumstances the lesson proceeds at a good fast clip.

This relatively good efficiency is due to three improvements made to NLS-SCHOLAR. The first improvement was to redesign and streamline the output routines, the ones which are responsible for producing English sentences out of information encoded in the semantic network. This resulted in a package that operates 5 times as fast as the old one.

The second improvement was to block-compile LISP-NLS. This technique provides a way of compiling several functions (LISP routines) into an entity called a block. Once a block is entered, function calls within it are very fast and variables' values are looked up directly, resulting in considerable execution speed-ups. It is not rare to see order of magnitude improvements from judicious use of this technique.

The third big improvement was to pre-compute the tasks' vectors. Previously, when a user's performance of a task was to be evaluated, the system used LISP-NLS to perform the

correct sequence of commands and to obtain the correct image of the work space. This was then compared with the result of the user's commands. In the present version of NLS-SCHOLAR, these correct images are obtained for each task at system generation time, and are stored away in a separate file.

A file handle is provided for each task, and is made accessible from the semantic network entry for the task so that the correct image can be retrieved from the file. Consequently, when a task is evaluated there is no CPU time wasted in generating the correct image.

SECTION III OPERATIONAL TESTING AND RESULTS

As described at the beginning of Section II, "operational" testing of successive versions of NLS-SCHOLAR started early in the course of our work. For this purpose we used BBN personnel ranging from completely naive users, through secretaries with experience using other computer-based text editors, up to experienced computer and behavioral scientists.

When our system was (reluctantly) pronounced ready, it was used in an informal but realistic testing environment by 14 non-BBN users. Among them were DOD personnel from the Air Force Data Services Center -- an outfit chosen by the Contracting Agency -- whose sophistication in using NLS ranged from very naive to experienced. In addition, the Contracting Agency solicited an independent evaluation from qualified Technical Personnel of the Information Sciences Institute (ISI) of the University of Southern California. The results of this evaluation are described in a report which is included in this document as an Appendix.

The data obtained from the operational testing is in the form of dribble protocols recording the "dialogue" between users and NLS-SCHOLAR. Over 50 protocols were of significant length (ranging from 20 to 90 minutes of on-line time) to be considered useful and to warrant their analysis. In addition to this data, an amount roughly equivalent was

obtained via our own internal testing using BBN's personnel. Taking everything into account, protocols representing approximately 100 hours of on-line time were analyzed. This amount of data is not sufficient to establish statistically valid results, but it is enough to sustain very definite qualitative conclusions about the system's capabilities and limitations.

General Results

The main thrust of this section is to describe and discuss a number of specific problems and problem areas identified in the course of the field testing. In order to frame the descriptions and to focus the discussions, we find it necessary, at the risk of being considered unscholarly, to present the general results of our analysis here rather than at the end of this section. They are:

1) The tutorial set-up appears to be very effective. New information is presented in bits and pieces of digestible size and users are kept on their toes (albeit in a very friendly environment) with dozens of questions they are asked to answer and NLS tasks they are asked to perform. Users <u>do</u> learn NLS: this is evident not only in the progress of their work, but also from personal communications (telephone calls, messages, and link ups).

2) The "supervised task environment", whereby the system evaluates the results of a user's performance of an NLS task and offers comments about it, appears to be very valuable. The system succeeds in pointing out mistakes and provides information useful for rectifying them. However, the system is sometimes over-zealous (rejecting outrightly the performance of a task for some trivial discrepancy) and sometimes fails to point out some erroneous action undertaken by the user. These shortcomings are not serious but they detract from the system's "intelligent" appearance.

3) A substantial part of the system's "smarts" resides in its English front end; NLS-SCHOLAR is designed so that the user can take the initiative anytime it is his turn to type and formulate requests (usually questions) to the system. Not surprisingly, however, this feature of NLS-SCHOLAR performed less satisfactorily than the rest of the system; only about 1/3 of the requests formulated were answered relevantly and usefully. This poor performance may have inhibited many users from asking more questions.

In view of the results outlined above, the rest of this section is concentrated on a detailed discussion of the performance of our English front end, and on the general issues it raises in the area of Natural Language

Comprehension.

Overview

Two points must be considered in order to view this last result in the proper perspective. In the first place, a large majority of the requests that the system failed to answer or answered incorrectly could have been handled satisfactorily with minor changes to the system and additions to its semantic network. Undetected spelling errors, unanticipated synonyms, common but not anticipated sentence syntax, lack of specific knowledge, etc., are examples of problems of this kind which are relatively easy to rectify as each one is found. As a whole, however, much time and effort must be expended to eradicate such problems entirely.

Secondly it must be borne in mind that the tutorial material is very clear and complete. It leaves relatively little room for doubt within the domain of procedural and conceptual knowledge that the question answering system is designed to handle. Consequently, the relatively few unanswered requests not covered in the "easy problems" category described above, reflected a combination of subtler doubts and the efforts of sophisticated users to concoct a question to assess the system's capabilities.

These questions remained unanswered either because they were expressed in round about ways (i.e., outside the set of paraphrases the system can recognize or had convoluted sentence structures), or because they were imprecisely formulated. The round about problem was not important in our case. It is more likely to occur in questions posed by users returning to the system after a partially forgotten previous exposure to its tutorial material. This situation could not develop within the period the system was tested.

Imprecisely formulated requests were much more common, within the relatively small number of hard-to-answer questions we are focussing on, than precise circumlocutionss. The relatively high frequency of imprecisely formulated requests and their inherent interest justifies the more detailed description and analysis of their nature which will be found later in this section.

The "easy problems"

Some examples of problems that are relatively easy to rectify are presented next:

Spelling errors - Consider for example,

"What is my current statement>?"
or "What does OK/C mean?"

In the first case, the system's spelling error correction list contained both the words "statement" and "statements", which resulted in "statement>" being corrected to "statements". The system knows what a "current statement" is (both the meaning of the concept and how to find out its present value), but it was hopelessly confused by "current statements". Given our current approach that emphasizes speed and expediency, the remedy is to eliminate "statements" from the spelling correction list. A better solution, such as performing morphological analysis and checking the agreement of verb and predicate numbers, would have required a fundamentally different approach.

In the second example, the system knows the meaning of most prompt symbols, and in particular that of the OK/C: prompt (notice the colon). While the system is prepared to accept many common abbreviations and misspellings of these symbols, OK/C was not anticipated.

Unanticipated synonyms - A very common group, exemplified by,

"Please review the one-character commands"
"How do I logout?"

"Explain the OK: prompt"

The system would have answered these requests correctly if they had contained the verbs "list" or "tell me about" or "give me" instead of "review"; "stop" or "quit" instead of "logout"; and "describe" instead of "explain". Fixing this may be trivially done by incorporating those verbs in the internal synonym lists of the system, or by incorporating their definitions and usages in the semantic network. Observe that "review" could have been used to mean something different from "list", e.g., to mean something like "Let´s go over the one-character commands again", after they have been presented once. To handle this kind of request is feasible but less trivial.

<u>Common but unanticipated syntax</u>

"What NLS commands are there?"

The almost equivalent question ("What are the NLS commands?") would have been answered correctly, but that format didn´t occurred to this particular user. (Neither did <u>we</u> anticipate the "are there" format!)

<u>Lack of knowledge</u> - A few examples are,

"How do I delete a file?"

"Tell me about control commands"

"What is an illegal delete?"

Deleting files was not possible from within NLS until very recently. This negative knowledge should have been incorporated in the semantic network, enabling the system to generate its canonical negative answer ("YOU MAY NOT DELETE AN NLS FILE USING AN NLS COMMAND"). This particular question generated a rather amusing dialogue (albeit very frustrating for the user) in which the system told him how to delete all sorts of things, except files. Also, although grouping certain commands under the category of "control commands" is certainly reasonable, the category was not anticipated and an entry for it is not contained in the semantic network. The same applies to "illegal delete".

Poor answers. - In some cases the answers generated by the question answering system were unsatisfactory. Examples are:

"Describe play"

PLAY IS THE COMMAND YOU TYPE TO THE TUTOR TO PLAY WITH NLS.

"What do you mean play with NLS?"

PLAY IS THE COMMAND YOU ---

"What is a field?"

A FIELD IS A SEQUENCE OF CONTIGUOUS DIGITS OR CONTIGUOUS LETTERS WITHIN A STATEMENT NUMBER.

"What does that mean?"

PLEASE REPHRASE YOUR REQUEST.

Obviously, giving more coherent and complete responses is something that can be taken care of. In the above case, since "field" is a particularly difficult entity to define in a more useful way, a specific example could be appended to the definition (FOR EXAMPLE, THE FIELDS OF 12A2 ARE 12, A, AND 2).

Unanticipated environments. - Many of our users worked on display terminals. Since NLS-SCHOLAR is geared for users of hard copy terminals, many of the requests originating in display environments were not contemplated in our design. For example, users of display terminals would often forget what task they were supposed to perform, or what question they were expected to answer. Every time the system reminded these users about their pending obligation, they would request a restatement of the question or task. Handling such requests ("What was the task?", "Ask the question again", "What did you want me to do?") is possible, but again, we did not anticipate them.

As can be inferred from the preceding descriptions, the common trait that unifies these classes of failures is our incomplete anticipation of what users would do in otherwise

expected situations. An elucidating comment to forestall our readers from reaching hasty conclusions is in order here.

It is very easy to disregard the importance of these "little failures" and attribute them solely to our lack of foresight. This condescending attitude, that can perhaps be subsumed as "How could they have forgotten X, or not taken Y into account?" fails to perceive the real issue. It is false to believe that incorporating X or bringing Y into the fold will make a substantial difference. The authors of this report did nothing else during the last 2 months of their work, and still the system is plagued with "little problems"! The crux of the matter, what must be recognized, is that when one is faced with the fantastic variety, the multitudinous aspects, and the changing modalities of the behavior of a human engaged in a dialogue with a machine, converging to a system relatively free of these "little problems" is a very long process. All we can say at this time is that this first round of field testing has been extremely useful in uncovering a large number of problems of this type, and that we expect the next round to uncover a smaller number.

The Harder Problems

We turn our attention now to the more interesting

failures of our English front end, those involving questions that were too imprecisely formulated for our system to answer. The imprecision of these questions stemmed from the anaphora they contained or, more seriously, from their "situational" character; that is to say, comprehending them would have involved understanding the <u>process</u> of the user's interactions with the system. These questions arose in such a form because the user assumed that the system was aware of the entire situation as it appeared to him; it is surprising to see how large an amount of contextual information must be taken into account before such questions can be properly understood.

The difficulty resides not so much in the literal interpretation of questions, as if they were precise formulations of the specific bits of knowledge the questioner might seek, but rather in figuring out what each particular person may have meant to ask, given his background, his previous experience, his previous performance, what he ought to know vs. what he seems to have learned, the environment he is working on, etc. These are very hard problems; they lie at the heart of the Natural Language Comprehension research area and their general solution still eludes us. Our purpose is to explain why these problems are so difficult, and to show the advisability of indirect solutions.

Many of these problems are rather subtle and it is easy to dismiss them because one can often stumble upon a seemingly general solution whose real underlying "ad-hoc" character is hard to perceive. To appreciate the difficulties involved, we shall see how a solution that seems satisfactory for a particular problem fails to apply to an apparently similar one. We shall proceed by analyzing five scenarios taken from our protocols. Each scenario comprises a description of a particular situation, the relevant context, and the question formulated. The scenarios are ranked in order of increasing difficulty, in terms of the mechanisms that have to be invoked in order to handle them.

Anaphoric reference

First scenario - The curtain rises after the student has been taught the purpose and usage of a fairly large number of "viewspecs" - characters used to specify how an NLS file is to be printed or viewed. Before leaving the subject, the system mentions several additional viewspecs, and then tells the user:

> As you can see, there are a great many viewspecs. If you are interested in what they control, you may ask me questions about them. However, the ones that have been introduced here are likely to be sufficient for most purposes.

At this point, the student asks:

"What do they control?"

This example is deceiving because it would appear that handling such a simple anaphoric reference is within the state of the art [Woods 1972]. The difficulty, however, resides in the lack of coupling between the question answering system and the tutorial material; in other words, the question answerer does not know the details of what the tutor has just finished teaching and cannot place the request in context.

A conceivable way to cope with this problem would be to have a complete internal representation (in the semantic network) of the tutorial material, and then synthesize the text the user reads from that internal representation. Given the present state of our knowledge on how to represent information in a semantic network and how to generate passable English from it, such an approach would fall short of our needs and would be totally inadequate for teaching naive users.

Another way to cope with this problem would be to re-write the text so that such anaphora would be inhibited from occurring, rather than being encouraged as they are in this example. The student is likely to frame his questions in terms of the words of the text, ("If you are interested

in _what they control_"), so the elimination of referential pronouns in the text might encourage him to eliminate them in his questions.

But even if we could synthesize the text gracefully from a semantic network or re-write it carefully with an eye towards forestalling anaphoric questions, other difficulties would arise as indicated by the next scenarios.

Elliptic structure

Second scenario - The system tells the student:

NLS FILES

In order to begin using NLS you will need to specify which ´file´ of information you want to work with.

> Each file is sort of like a notebook or folder in which you can keep information.
>
> You may keep as many different notebooks (files) as you like.

Files are automatically stored when you are not using them.

Before you can work with a file you must ´load´ it from the storage into the working space of the computer.

Each file has a name so you may refer to it easily.

> File names are made up of letters and digits and may be quite long - like BUDGETFORFISCAL75.
>
> No distinction is made within file names between upper and lower case letters - both are treated as the same character.

At this point, the student asks:

"What about blanks and other special characters?"

(Before going on to point out the new problems inherent in this example, we should mention in passing that, here again, the occurrence of this question could have been prevented by re-writing the text so that it specified in exact detail what characters could be used in designating file names. This would provide, however, more detail than most users really want and is the sort of information that belongs more properly in a reference manual than in a tutorial.)

Let us ignore the problem of the conjunctional form of the question, which we are presently unable to handle, and simplify it to be

"What about blanks?"

The new problems that face us here are the elliptical form of the question (it´s not a sentence) and the multiplicity of logically acceptable referrents. For example, focusing on the last sentence uttered by the "tutor", the answer would be

> UPPER CASE BLANKS ARE TREATED THE SAME AS LOWER CASE BLANKS.

If instead one focused on files (rather than on file names), one might generate the answer

> YOU MAY KEEP BLANKS (as well as other special characters) IN FILES.

In order to generate the answer that the student is actually seeking, i.e.,

FILE NAMES MAY NOT CONTAIN BLANKS

we need a crucially important new component: a model of the user.

Such a model would be used, perhaps unconsciously, by a human tutor in answering this question. An experienced tutor knows that the rules about permissible characters in file names vary from system to system and might be expecting such an enquiry about file names from a non-naive student. The fact that this student chose the term "special characters", not mentioned in the text, indicates that he has some previous experience. He certainly wouldn´t be asking whether blanks could be stored in files, or imagine that blanks come in both upper and lower case varieties. Thus for a system to cope with a question like this, it would need to have a broader knowledge base than that describing NLS; it would need to have knowledge about the capabilities and expectations of the user.

Indeterminate Reference

Third Scenario - A similar situation (but with an interesting twist) appears next.

Anticipating students´ uneasiness and nervousness

before performing their first task, NLS-SCHOLAR gives them rather precise instructions. To wit:

LOADING A FILE

I´d like to show you the file named DINNER so you can see how an NLS file is structured.

Your first task is to load this file so you can work with it.

When the BASE C: appears, type the command

load <SP> file <SP> DINNER <CR>

Note that you should terminate each word of the command with a space (<SP>); you should terminate the entire command with a carriage return (<CR>).

(You may type DINNER in either upper or lower case letters.)

As this single command completes the task, when the next BASE C: appears type

quit <SP> <CR>

I´ll then check what you´ve done and point out any mistakes you may have made. Please be sure you type a <SP> after "quit", before you type the <CR>.

If you make a typing error while doing this task, you may use <CTRL-A> to remove the last character, or <CTRL-X> to delete the entire line.

These commands work in the same way whether you´re typing to me or to NLS.

Do you have any questions before doing this task?

And here the user asks:

"Do I type the entire command?"

This is a situation in which even a human tutor might have difficulty figuring out what this user wants to know. Let´s consider some of the possible answers.

1) Focusing on the last two sentences before questions are invited, the system could reply

```
NO.  YOU DELETE THE LAST CHARACTER USING
    THE <CTRL-A> COMMAND.
    PROCEDURE: YOU DEPRESS THE CTRL
    KEY AND THE A KEY SIMULTANEOUSLY.
```

   This is NLS-SCHOLARese for "No. You don´t have to spell out <CTRL-A> to delete a character. You only have to depress the CTRL key and the A key simultaneously."

2) Focusing on the third and the fourth sentences, the system could answer with something akin to "Yes. You must spell out the entire command exactly as you are told."

3) Finally, the answer could be directed to the fact that all parts of a command must be specified, and to type only the first part of a two part command leads nowhere.

The twist is that the user model in most people´s minds would not be sufficient to identify the purpose of the question. Why would anyone ask it? Indeed, isn´t the manner in which the commands for the first task are to be typed clearly described? Isn´t it self-evident that all parts of a command must be specified before it can be executed? And

haven´t students already used the <CTRL-A> command in the introduction?

The solution to this riddle is that this particular questioner was familiar with NLS and was accustomed to typing just the first letter (or two) of each command, using NLS´s expert input mode. His question reflected his doubt that NLS-SCHOLAR really meant for him to type each and every character of a command, and wanted the system to confirm its instructions. This familiarity can be gleaned from watching his performance on <u>subsequent</u> tasks, but not at the time the question was asked, just before the first task in Lesson One!

It might be argued that the needed information could have been obtained from a user profile collected beforehand. The problem of acquiring it might be handled by inserting questions into the introduction about his previous experiences. One could find out, for example, whether he was familiar with terminals, computer systems (if so, whether TENEX or others), editors (if so, whether NLS or others, and if NLS, which version), etc. If his answers warranted it, certain parts of the introduction might be skipped; a fairly detailed user profile could be generated from this information.

A limited user profile could be easily gathered and should be of assistance in coping with questions like the

above, but using it in the way we have described implies that the requisite knowledge about other computer systems, terminals, characteristics of user behavior, etc. will all have to be within the system's knowledge domain. This multifold expansion of the system's field of expertise and its integration into a coherent whole, would be a formidable undertaking.

It may be argued that the adjective "entire", appearing in the fifth sentence of the tutorial material and in the question, is a clue that helps to link the question with the desired answer. As mentioned earlier in another context, a person involved in a dialogue often adopts the same words that were just used by the other party. Here then, we have a possible way out: lexical clues can help disambiguate what a student's question is about. But that won't help us sufficiently as the next scenario will show.

<u>Fourth scenario</u> - After having learned how to use the Delete command, and after having actually practiced the command by deleting three statements in his own working file, the user is told:

> Please print the modified DINNER file so you can see that the statements containing "tomato", "rhubarb", and "strawberry shortcake", have all been deleted from the DINNER file.

After he prints what he is asked, the system continues with:

> Note how the statement numbers have been changed by NLS. You can see that many statements have been renumbered (´promoted´), some of them acquiring the statement numbers of the deleted statements.
>
>> Although statements 1A, 3A1, and 3B were all deleted, these statement numbers still exist in our file -- but the statement contents are now different.
>
> Would you like to ask any questions?

At this point, the user asks:

"Can I delete these modifications?"

Since many people find it hard to understand this question, let us clarify it with the help of a paraphrase

> "Can I restore the contents of the file to what they were before anything was deleted?"

Several new problem elements are introduced into the picture by this scenario.

In the first place, the anaphoric reference is to previous actions undertaken by the student (or on the student´s behalf) using NLS. The reference is directed neither to concepts explained earlier, nor to anything represented in the semantic network (the question is <u>not</u> "Can I delete modifications"). This illustrates the need to

bring into focus the history of changes (modifications) made to the user´s work file, which is not hard to do in our system.

In the second place, here we have a case where "modifications" could be misconstrued as being inspired by "the modified DINNER file" in the tutorial material. In reality, "modifications" for this user turns out to have a much firmer root: experienced NLS users know about the "modification file" (a file where all the changes made to a working file are kept until the working file itself is updated) and how to manipulate them. This user is not naive: he knows that NLS provides specific ways of "undeleting" and he is simply and benevolently testing how much NLS-SCHOLAR knows about them.

In the third place, we have the rather incongruous use of the verb "delete" with the object "modifications". All that the student has learned up to this point indicates that "deleting" is a positive action resulting in something being eliminated from his work file, but here deleting something would result in the reappearance of that which was deleted earlier! If we know what kind of "modifications" the student is talking about, we can make sense out of the question without too much regard to the verb used (try, for example, "restore" or "undelete", or "do something about"). Therefore, here we have a case where what the student <u>must</u>

be speaking about outweighs other interpretations stemming from his choice of words, such as "Can I delete (the statements containing) these modifications?"

Fifth Scenario - We begin at a point where the system has just taught the student how to load and print a particular file, and the student has successfully performed two tasks requiring him to perform these actions. The student then has available the following printout of the contents of the file.

```
< TUTOR, DINNER.LNLS;1, >, 14-SEPT-75 13:43 LAC ;;;;
  1 SOUPS
     1A tomato
     1B vegetable
     1c cream of mushroom
  2 ENTREES
     2A fried chicken
     2B prime ribs
     2C scallops
        2C1 broiled
        2C2 fried
     2D salmon
        2D1 with cream sauce
  3 DESSERTS
     3A pie
        3A1 rhubarb
        3A2 blueberry
     3B strawberry shortcake
     3C ice cream
        3C1 blueberry
        3C2 maplenut
        3C3 chocolate
        3C4 coffee
        3C5 peppermint
        3C6 cherry
```

The system begins to describe this file as follows:

THE ORIGIN STATEMENT

Let´s look at the information in the file.

Notice that there is a line at the top which gives identifying information about the file

This line is called the ´origin statement´ and is supplied by NLS.

First it gives you the name of the ´directory´ (a place in the memory) in which this file was stored. Then it gives the full name of the file, and the date and time of its creation.

The file name includes an ´extension´ specifying what kind of file it is.

In this case it says that this is an "LNLS" file. (LNLS stands for LISP-NLS and indicates that this file was made by our LISP implementation of NLS.)

The number after the file name is called the ´version number´.

The "1" here indicates that this is the first version of the DINNER file that´s been made.

Do you have any questions?

And the student asks:

"Are the brackets part of the statement?"

Here we have two anaphoric references ("the statement" and "the brackets") and a questioned inclusion relationship between them.

Finding the referrents (the first line of the printout as a realization of "the statement", and the left and right angle brackets within it as "the brackets") involves methods of solution not required previously. "The statement" can

readily be assigned the referrent "origin statement" by means of the previously hypothesized representation of the tutorial material and by focusing, but from there on we face entirely new problems. In the first place, the student uses "the brackets" to describe some portion of the <u>content</u> of a statement. Surely we can not expect the system to be capable, in general, of dealing semantically with the contents of user files. In fact, referring to statement 2D as "the fish" is possible only because of our knowledge of zoology, which has little to do with text editing systems or with NLS in particular.

Secondly, although "origin statement" is a perfectly valid referrent for "the statement", what is really meant is "the particular realization of an origin statement that is represented in the first line of the print out". Presumably, quite a bit of inconclusive inferencing will have to go on before the system quits trying to find a connection between brackets and the concept of an origin statement (after all, <u>square</u> brackets <u>can</u> be used in file names!)

In the third place, even after the correct referrents have been identified, what sense does the question have? Why shouldn´t a part of the content of a statement not be a part of the statement? Isn´t this obvious? And if so, why would such a question be asked? If the interpretation "upper case

blanks are treated the same as lower case blanks" could be rejected for being trivial why can't this one be rejected similarly?

The truth is that we don't know why this particular user asked the question. We can only speculate that he was a TENEX user and was wondering if the angle brackets were used in a fashion similar to the way directory names are denoted in TENEX; or he may have been prompted to ask this question because of the way NLS-SCHOLAR denotes certain keys (<CR>, <CTRL-A>, etc.

This is a good place at which to stop and recapitulate the preceding analyses and discussions. We have seen how each scenario has introduced new problems, and how each new problem has required more and more complex methods of solution -- and yet, there is no indication that this escalation of complexity has ceased.

Proposing those methods, we stretched available ones and hypothesized new ones to such an extent that continuing to do so would have been utterly unrealistic. For example, the user models we require would have to encompass a large amount of "world knowledge" in order to cope with situations such as the ones exemplified in our scenarios, and yet the theory underlying such models is in its infancy at best.

The exercise we engaged in is certainly useful and

illustrates the need for continuing research, but above all it demonstrates the need for a pragmatic approach, i.e., one based on accepting the seriousness of the difficulties and finding a way around them. Rather than exploring a large number of plausible interpretations of a user's request, it is better to either forestall the request, or to seek its clarification.

SECTION IV - RECOMMENDATIONS AND CONCLUSIONS

In this section we summarize conclusions reached for the most part in previous sections, and we formulate recommendations for further work. Our contention is that one more year of relatively low level effort can make NLS-SCHOLAR a very useful operational system.

Our first recommendation is to continue to improve the English front end module to rid it of the nagging little problems described extensively in the preceding section. This can only be done on a continuing basis, correcting the problems as they appear in the course of bona fide usage of the system by the type of users for whom it is intended. This process will be long, but the result should be a system able to answer as many as 80% of the requests posed. In parallel with this effort, techniques such as the ones sketched in the previous section for circumventing the harder problems should be developed and tested, and research efforts aimed at attacking these problems head-on should be stepped up.

Our second recommendation is to improve the task evaluation module in the following ways:

1) Make it point out more clearly what is wrong with a student's result. For example, when this module responds

"I wanted you to change A into B but you changed A into C"

it is hard to see sometimes what the difference between B and C is. In other words, in our efforts to avoid presenting the offending text in isolation without contextual information, we went too much in the other direction; we showed so much of the surroundings that the specifics got drowned!

2) Augment the existing task entries in the semantic network with a list of expected errors and specific ways to report them. This would permit by-passing the standard reporting format if one of these specific errors were found.

3) Implement a "let me fix it" facility to avoid the sometimes costly consequences of the task evaluators's zeal. This facility will hand back a task environment to the user after the system has found fault with it and has required the user to do it all over again. In this way, users that realize what is wrong and what is expected of them could patch up their work and satisfy the task evaluator's requirements in their own way.

Finally, what would really make this module "intelligent" would be to give it the ability to understand and interpret the user's intentions and to offer helpful comments. It is not enough to point out what is wrong with a <u>result</u>; the most helpful situation is one where the user's <u>solution</u> methods are scrutinized and criticized. This area

is certainly one where further development is needed.

Our third recommendation addresses the tutorial material. Although it is certainly in good shape, it could be improved by adding the capability for the user to redirect the order of presentation of a lesson via requests such as:

"Let's go back to DELETING BRANCHES"

"Tell me again about <CTRL-X>"

We have the necessary groundwork to handle these requests for review. The only problem is how to apprise the user of the new context he is to work on after his request has been fulfilled; that is, how to indicate gracefully that his file has been restored to an appropriate earlier incarnation.

We could also handle requests like

"Let's skip this task"

without too much difficulty. Here the necessary changes to the user's file, to bring it to the state it would have acquired after the task had been completed, must be explained and justified. Requests of the form

"Let's skip all about INSERTING"

and

"Teach me about VIEWSPECS" (implying a large forward jump)

raise other issues as well. Not only is the problem of bringing the file up to date more complex to explain as many

tasks may be involved, but also some of the concepts and terminology skipped over may be needed by the student in order to comprehend the following material. Allowing the student to review is relatively easy; allowing him to skip forward is quite difficult given the linear development of the textual material.

Epilogue

It is easy to jump to the conclusion that the unresolved problems we have dealt with so extensively, preclude systems such as NLS-SCHOLAR from becoming useful in an operational environment. This conclusion would be erroneous for several reasons:

a) The frequency of occurrence of "hard problems" is very small. Most of the users' requests we have seen belong to the "easy to answer" category, regardless of the actual performance of the present version of NLS-SCHOLAR.

2) As more and more of the little problems are ironed out, users will be positively reinforced towards expressing their requests in the kind of English the system understands, and with the precision of formulation the system requires.

c) As the number of failures decreases and the number of users increases, it becomes both feasible and economical to provide a human expert to back up the system as a kind of "consultant". In a computer network environment, many

users from different sites could take advantage of this immediate and most effective form of help. Notice also that while human expertise is concentrated in the hands of one expert at any one time, experts located in many sites can take turns at minding the system; i.e., human expertise may be concentrated but not centralized.

Waiting until "intelligent" CAI systems become capable of 100% stand alone operation is both futile and counterproductive. It is futile because that kind of performance is probably impossible to obtain (just think of how few people can do it!). But, more importantly, it is counterproductive because widespread use of an 80% effective facility, for example, would multiply by a very large factor the consulting capacity of a human expert, enabling him to reach more people than he could otherwise and to address himself to the relevant problems quickly.

APPENDIX

Review of NLS-SCHOLAR by ISI

The following evaluation report was written by David Wilczynski, of the Information Sciences Institute of the University of Southern California, at the specific request of the Contracting Agency.

I. INTRODUCTION

This review is based on my own experience in early August 1975 with NLS-Scholar, a mixed-initiative tutorial CAI system for teaching a basic subset of the text editing subsystem of SRI´s NLS programming system.

NLS-Scholar, programmed in INTERLISP, was written by Mario Grignetti and his group at BBN. The system has evolved from Jaime Carbonell´s Scholar (which teaches South American geography) together with substantial influence from Brown´s SOPHIE system. The system is organized to:

a) Present textual, tutorial material to introduce the user to a terminal and to NLS.

b) Provide a simulated NLS system to the user on which to practice what he has learned, as well as to do system-generated NLS tasks.

c) Provide a natural language question-answering component which responds to user queries by: 1) doing AI-like searches in its fixed data base, or 2) "executing" the right NLS commands on the user´s current file to answer dynamic questions.

d) Present various NLS tasks to the user to test comprehension of the material just presented.

The course is divided into the following lessons. Each lesson takes about 1 hour, with many variables determining the exact length, load average, attention span, competence, etc.

Introduction - Control characters

Lesson 1 -

Commands: Load File, Print File, Delete, Update

Concepts: NLS files, NLS commands, NLS prompts, structure units (statement, branch), string units

Lesson 2 -

Commands: Print Rest, Jump, one-character commands

Concepts: Control Marker, content addressing

Lesson 3 -

Commands: Insert, Create File, Substitute

Concepts: Level, level adjustment

Lesson 4 -

Commands: Print, Transpose, Move, Copy

Concepts: Plexes, Groups

Lesson 5 -

Commands: Show Viewspecs, Set Viewspecs, Reset Viewspecs, Output, and Help.

Concepts: Viewspecs, Text File

II. GENERAL IMPRESSIONS

NLS is well suited for CAI methods; NLS concepts are short, factual, and "nonphilosophic," a good method is available for testing competence (either interfacing directly to NLS, or simulating it), and the information is incremental and additive rather than diffuse.

The main point is Scholar did teach me NLS. At the start of the program I knew nothing about NLS other than what it is; now I know the NLS terminology and how to use the system. However, improvement is necessary in several areas if Scholar is to be a finished production program, competitive with possible alternative teaching methods. The following two sections will review Scholar's strengths and weaknesses.

III. THE ENVIRONMENT OFFERED BY SCHOLAR

The Scholar CAI system is classical in that text is presented to the student in prearranged frames with tests usually following each. The inclusion of a natural language interpreter is an innovation which allows the student to ask

questions during the program. It turns out that this mode of operation has advantages for nonstudent types. Studies have shown that people relate well to computers, suffer less anxiety, and feel freer to experiment and ask questions in a CAI environment. The critical aspect of such a system is its transparency.

If the student notices (or becomes preoccupied by) the CAI machinery, he can perform in the short term (answer questions, do short tasks), but lacks global comprehension. Thus the type of display and the "smoothness" of the system become important factors for people not used to operating such devices. Specifically, NLS-Scholar is intended for typewriter terminals. Having written a CAI system for such terminals myself, I have verified that all students are acutely aware of the typing noise and slow speed. I used NLS-Scholar on a 2400 baud video terminal and was much more satisfied with the results. Since there are times when hardcopy is needed for back referrals, BBN would do well to offer the appropriate hardcopy text to the student as an addition for the video terminal.

A parameter of system smoothness is its responsiveness. A high load average (virtually anything above 4) combined with the slowness of INTERLISP made Scholar move at an unbearable crawl. When the load average came down to 1 or less, the system moved about sprightly. The difference here is more than one of convenience. No user (unless he is forced or paid) will sit through a session of Scholar on a machine with a high load average. If he must, it will turn out to be a painful, wasteful way to learn NLS.

A few of the INTERLISP features caused some unnecessary distractions. I found the garbage collect messages ("Excuse me, while I rearrange my memory!") disconcerting since they caused a visual break in my concentration. I appreciate the attempt to explain the impending delay, but I think the typed message is too visible.

The preprocessing of all questions and responses by DWIM also caused some amusing incidents. For example, in answering the question, "What character prints the context of the CM?", I responded "/". DWIM turned the slash into a "?" (a common INTERLISP occurrence) and then NLS-Scholar told me that "/", not "?" was the correct answer. Those sort of bugs are not serious and easily repairable, but must not exist in a released product.

NLS-Scholar offers a medium which can be started when desired (assuming machine availability), stopping at arbitrary points, and proceeding in whatever pace is comfortable. If NLS-Scholar were set up to operate at different modes (beginner, expert, review) then the problem

of retraining and refreshing previous NLS users would be simplified. This may not be a simple addition to make in NLS-Scholar, but judging from discussions with users at Gunter AFS, it would be powerful and useful.

IV. TEACHING COMPONENTS OF NLS-SCHOLAR

The three main components of NLS-Scholar are: a) the tutorial information, b) the natural language interpreter, and c) the test management. The first and third are CAI standards, while the second is in the realm of Artificial Intelligence (AI).

A) The text material was impressive; it was presented concisely and accurately. At no time did I feel that I was being either overloaded or nursed through, both factors which led to effective and willing comprehension of the material. It is easy to overlook or underestimate quality in this area because good tutorial services are not as visible as poor ones. Because of this phenomenon I want to emphasize the excellence of the tutorial information.

B) The natural language interpreter is more complicated to evaluate. Most likely, it is the most complex part of the system, yet probably the least useful in its present form. The main problem concerns its robustness. Often I asked a simple question like "Please review the one-character commands," and got only a "Please rephrase your request" reply. In this case I think the problem is that "review" is not part of Scholar's dictionary. However, in rephrasing the question to something like "Tell me about the one-character commands," I would just get a list of them without functional definitions. To get what I wanted I would have to ask for each individually (e.g., "Tell me about the command."). It is disconcerting to have the parser or retrieval mechanism fail on a simple request, but not to know why is worse. Just asking for a rephrase does not indicate what the failure was; this information will surely be useful in composing a different request. Whether most users would want such information is a different question; I would have liked it.

It is hard to be critical of this natural language business, since the problem is still a major research, not developmental issue. Still, I wonder if Scholar's interpreter is state of the art; I am thinking of Woods' moon rock program. Since that program is also a BBN product, it would be interesting to get a comparison of the two systems from the NLS-Scholar group.

The lack of robustness of the English interpreter detracts somewhat from Scholar; I found myself not using that component. The table look-up kind of questions it could answer would be better solved by just having access to the table in some primer format. Again, the lack of field testing may indicate that this is just a personal reaction; but the shallow range of questions and answers makes the current worth of this subsystem suspect. Certainly, it doesn't fulfill the capabilities of a human tutor.

C) The test management phase of Scholar is composed of a series of questions which are answered either by doing an NLS task, or talking directly to the Scholar top-level. In both cases the answers or performance are evaluated with feedback as to correctness. The ability to check answers is one of the more difficult tasks for a CAI system when the domain of true-false or multiple choice questions is not used. Scholar does admirably here but is far from perfect.

The top-level type questions, (e.g., "What is the statement number of the statement that will be printed if I now use the backslash command?") will be looked at later. The NLS tasks, the heart of the testing component, will be reviewed in depth.

The basic mechanism for matching a task answer to the correct one seems to be:

a) If a file manipulation task is involved (e.g., INSERT, DELETE), then the resulting file and the CM (control marker) are checked against Scholar's expectation.

b) If a printing task is involved, the output of the print command is trapped and matched against expected print, and the CM is checked for positioning.

At no time does it appear that Scholar looks at the student's input sequence. This lack leads to many unfortunate experiences. For example, one task asked to delete two consecutive statements, expecting the user to use the sequence, "delete statement 1B5, delete statement 1B5," to account for the renumbering done by NLS. I tried, "delete statement 1B6, delete statement 1B5," to accomplish the same effect. Scholar told me I did the task correctly and then the next frame described how I could have accomplished the same task by deleting statement 1B6 and then deleting statement 1B5. Not serious, but the question of system transparency arises.

A more serious flaw in this purely extensional form of testing appeared in the task to test the use of ´CTRL-E> for inserting a series of statements. I did the task by inserting all the statements at the same level (superfluously using CTRL-E after each insert) before going back to insert substatements. Even though the resulting file was correct, the CM was not where Scholar expected it and so I was informed of this "error" and told to redo the entire task from scratch! Needless to say, I didn´t enjoy retyping the whole thing. Worse, however, was the failure of Scholar to recognize what I did, tell me the right way to do the task (i.e., use one CTRL-E and move up and down levels using the L: prompt) and then let me proceed. It is, however, easier to be critical of this flaw than to suggest an alternative. A deep understanding of the intensional command strings represents a large (perhaps unknown) effort. If accomplished, there is no question that the system will appear much more intelligent then it currently does, as well as being more useful.

Other examples of situations where this type of problem come up can be given, but are not necessary to this review. Some of the techniques used to check top-level questions (those not requiring the NLS simulator) are also open to improvement. For example, one question expected CTRL-X as the answer; I typed <CONTROL-X> and was told I was wrong. Another time I answered a question with LINE-FEED and Scholar wanted <LF>. These two cases should not be construed as nitpicking, but as an attempt to point out situations which make Scholar seem less suitable as a training method than standard teaching methods. Too many of these trivial flaws will discourage the CAI user.

V. SUMMARY AND CONCLUSIONS

As I mentioned before, it is much easier to point out flaws in a CAI system then to recognize its qualities. Experience with standard methods give rise to expectations which are then used to judge CAI systems. Yet, criticisms of Scholar should be tempered by one observation, Scholar does teach the student NLS effectively. Assuming that the local bugs in Scholar are fixed (a few have been described in this paper), a useful system exists which can be used to train potential NLS users.

Still, changes can be made which might expand its range of use as well as improve its performance. Several have been pointed out in this paper, for example, making the natural language component more robust, adding analysis of the user´s input to the current extensional analysis, endowing Scholar with other training modes, expert, review, etc.

None of these possibilities are simple; more field testing is necessary before firm conclusions can be made one way or the other. Yet, once Scholar is made more complete in its coverage of NLS, it will be a viable product and should be evaluated as such by agencies interested in NLS.

Some purely system questions also need addressing. Can NLS-Scholar be a viable product as an INTERLISP program (thus bound to TENEX)? Are there enough machines with enough time slots of low load average to accommodate the potential Scholar users? I am sure other questions of this type will arise if research into Scholar is continued.

Comments on the review

by Mario C. Grignetti

It seems to me that the review is, overall, a rather positive one. NLS-SCHOLAR seems to be able to do its most important job, i.e., teach NLS.

Many of the problems that Dave points out are trivial to take care of: garbage collection messages, DWIM's busy-bodiness in unwarrantedly exchanging "/" for "?", and more ways to represent CTRL-X or <LF> than we anticipated. After all, the main goal of the field testing performed under this contract was precisely to bring up these kinds of problems.

Dave is wrong in his assertion that "at no time does it appear that Scholar looks at the student's input sequence": The system does look at the student's actual input when he answers questions. The fact that Dave's clever answer (delete statement 1B6 and then delete statement 1B5) was not handled intelligently was due to a stupid bug in one of the predicate functions in our answer evaluation module. Again, this is a case in point for the usefulness of this type of testing to the system's designers. In general, however, Dave's criticism is valid: when the student performs a task using NLS, the commands he types are not looked at and only their consequences are used to evaluate what he has done. We'd like very much to tackle the difficult problem of

intentional comprehension; if solved we would have a much smarter system!

Other difficulties referred to in the review are more serious. Indeed, we need to provide feedback as to why a request fails to be understood. We had wanted to tackle the problem of partial comprehension and try a few strategies that appear promising. However, the pressures arising from limited time and resources, and the purely developmental type of work in which we have had to confine our efforts, precluded the performance of sorely needed research work. With respect to our use of "Wood's moon rock program", this is another thing we've kept on the back burner for some time. However, it is questionable that just a more powerful parser would have made a lot of difference in the system's ability to respond to student's requests. The difficulty here resides not so much in the literal interpretation of questions as if they were precise formulations of the specific bits of knowledge the questioner seeks, but rather in figuring out what each particular student may have meant to ask, given where he is, his previous performance, what he ought to know, what he seems to have learned, etc., etc. It is surprising to see how many questions are unanswerable, even to a human, when taken in relative isolation.

Finally, a word about efficiency. We do not think that 3 CPU minutes per hour is a terribly inefficient and

unacceptable way to administer a CAI lesson. We agree however (and wholeheartedly!) with Dave's observation that when the load average in a general purpose time-sharing system such as TENEX reaches about 4, it is better to quit and go home. This is not a problem that affects NLS-SCHOLAR alone; when a large system such as TENEX is saturated, nobody gets anything done efficiently, including NLS users.

REFERENCES

[1] Grignetti, M C., Hausmann, C. and Gould, L. "An ´intelligent´ on-line assistant and tutor - NLS-SCHOLAR," National Computer Conference, 1975.

[2] Grignetti, M.C., Gould, L., Bell, A.G., Hausmann, C.L., Harris, G. and Passafiume, J.J. "Mixed-Initiative Tutorial System to Aid Users of the On-Line System (NLS)," ESD-TR-75-58, AD A007 828, November 1974.

[3] Bobrow, D.G. and Wegbreit, B. "A Model and Stack Implementation of Multiple Environments," Communications of the ACM, Vol. 16, No. 10, October 1973.

[4] Teitelman, W., et al, Interlisp Reference Manual, Bolt Beranek and Newman and Xerox Corporation, 1974.

[5] Burton, R.R. "A Semantically Centered Parsing System for Mixed-Initiative CAI Systems," paper presented at the Association for Computational Linguistics Conference, Amherst, Massachusetts, July 1974.

[6] Brown, J.S. and Burton, R.R. "Multiple Representations of Knowledge for Tutorial Reasoning," Representation and Understanding: Studies in Cognitive Science. Editors D. Bobrow and A. Collins, Academic Press 1975.

[7] Woods, W.A., Kaplan, R. and Nash-Webber, B. "The LUNAR Sciences Natural Language Information System," Final Report, BBN #2378, June 1972.

[8] Fillmore, C.J. "The Case for Case," in Universals in Linguistic Theory, (eds.) Bach and Harms, Holt, Rinehart and Winston, 1968.